Efficient and Good Delaunay Meshes From Random Points

Mohamed S. Ebeida [a] Scott A. Mitchell [a] Andrew A. Davidson [b] Anjul Patney [b] Patrick M. Knupp [a] John D. Owens [b]

[a] *Computing Research, P.O. Box 5800, Sandia National Laboratories, Albuquerque, NM 87185-1318*

[b] *Electrical and Computer Engineering Department, University of California, Davis, CA 95616*

Abstract

We present a Conforming Delaunay Triangulation (CDT) algorithm based on maximal Poisson disk sampling. Points are *unbiased*, meaning the probability of introducing a vertex in a disk-free subregion is proportional to its area, except in a neighborhood of the domain boundary. In contrast, Delaunay refinement CDT algorithms place points dependent on the geometry of empty circles in intermediate triangulations, usually near the circle centers. Unconstrained angles in our mesh are between 30° and 120°, matching some biased CDT methods. Points are placed on the boundary using a one-dimensional maximal Poisson disk sampling. Any triangulation method producing angles bounded away from 0° and 180° must have some bias near the domain boundary to avoid placing vertices infinitesimally close to the boundary.

Random meshes are preferred for some simulations, such as fracture simulations where cracks must follow mesh edges, because deterministic meshes may introduce non-physical phenomena. An ensemble of random meshes aids simulation validation. Poisson-disk triangulations also avoid some graphics rendering artifacts, and have the blue-noise property.

We mesh two-dimensional domains that may be non-convex with holes, required points, and multiple regions in contact. Our algorithm is also fast and uses little memory. We have recently developed a method for generating a maximal Poisson distribution of n output points, where $n = \Theta(\text{Area}/r^2)$ and r is the sampling radius. It takes $O(n)$ memory and $O(n \log n)$ expected time; in practice the time is nearly linear. This, or a similar subroutine, generates our random points. Except for this subroutine, we provably use $O(n)$ time and space. The subroutine gives the location of points in a square background mesh. Given this, the neighborhood of each point can be meshed independently in constant time. These features facilitate parallel and GPU implementations. Our implementation works well in practice as illustrated by several examples and comparison to Triangle.

Key words: Computer-aided design, engineering, and manufacturing; Computational geometry and topology; Product and assembly modeling; Geophysical applications; Mesh generation

1. Introduction

Many applications in computational geometry begin by constructing a Delaunay triangulation of a set of points scattered in a given domain. Triangles in a Delaunay triangulation have circumcircles that do not contain any other vertices, and have desirable geometric shape. If the domain is non-convex or contains internal edges, the triangulation must respect the boundaries of the domain. Constrained Delaunay triangulations contain the required edges, and a

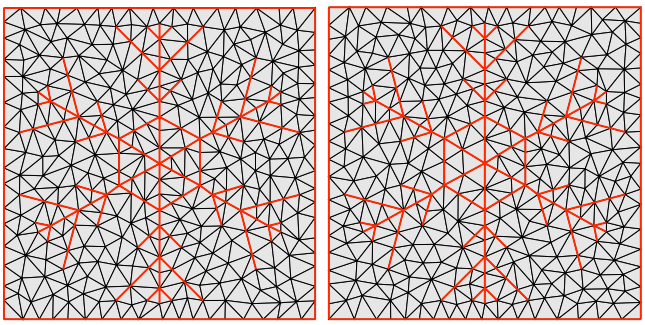


Fig. 1. Two random CDTs with the same radius and domain.

Email address: msebeid@sandia.gov (Mohamed S. Ebeida).

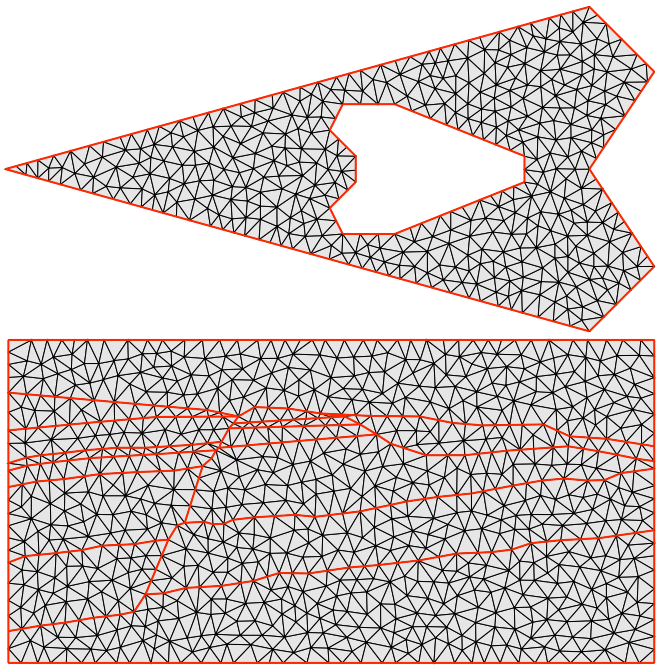


Fig. 2. Top, a non-convex fracture domain with a hole. Bottom, a seismic domain; our implementation succeeded despite the user selecting a coarser mesh size than the theory requires.

triangle's circumcircle contains no point visible to the triangle's vertices. Covering triangulations [28] add interior points to improve triangle angles, but constraint edges and vertices limit the improvement. In a Conforming Delaunay Triangulation (CDT), constraint edges are subdivided as well, greatly improving mesh quality. Each constraint edge is a union of triangle edges, and triangles are constrained Delaunay. CDT is important in many fields such as interpolation, rendering, and mesh generation. Well-shaped meshes of well-spaced points have many useful properties [27].

A very effective family of CDT algorithms is based on Delaunay refinement: start with a coarse mesh, and insert a point at the center of large Delaunay circumcircles. We contrast and bridge our method to the root of this family's tree, Chew [8]. Since Chew's seminal paper, Delaunay refinement has been generalized in many ways. The most relevant generalization for us is that new points do not need to be at the exact center of a Delaunay circle; indeed our work shows they can be placed randomly, as long as they are far enough away from prior points. Off-centers [37] inserts a point between the center and a short edge; it reduces the total number of inserted points by implicitly grading the mesh size. It also improves numerical stability. In three-dimensions, nearly-planar tetrahedra can be avoided by perturbing points. This can be done randomly [9] or deterministically [14]. This can be done symbolically or with actual coordinates or the Voronoi weights [6]. Randomly inserting a point, say within a smaller circle concentric with the Delaunay circle, reduces the bias.

Parallel Delaunay refinement is possible. The points used to fix different simplices will interfere with one another, but this can be resolved by only inserting the non-conflicting points, and taking multiple passes [34].

In any event, Delaunay refinement inserts *biased* points; an *unbiased* process selects a new point outside the (constant) radius r disk of any other point, but is otherwise chosen uniformly at random from the remaining disk-free area of the domain. This is also known in spatial statistics [4] as the hard-core Strauss disc processes with inhibition distance r_1 and disc radius r_2, where for us $r_1 = r_2$. The limiting distribution is called a maximal Poisson-disk sample (MPS) in graphics.

The probability of inserting a point at a given location is independent of the location. For Delaunay refinement the insertion probability depends on intermediate properties of the algorithms, such as the order in which bad-angle triangles are addressed and the DT angles and circle centers. The bias may be difficult to understand, describe, or predict, although spectrum analysis of pairwise distances can measure bias. Unbiased points have spectra with the "blue noise" property. Unbiased sampling algorithms have a long history in computer graphics relating to image synthesis, including applications in anti-aliasing [22] and Monte Carlo methods for ray tracing, path tracing, and radiosity [38].

Random meshes are useful in several contexts. The effects of mesh structure on modeling fracture in solid mechanics was studied in detail in the 1990's; see Bolander and Saito [3] for a thorough discussion. For some finite element methods, crack propagation is limited to triangle edges, or dual Voronoi cell edges. Structure also plays a role for spring networks, e.g. crack formation may depend on the orientation of the mesh with respect to the stress field. In either method, the locations of fractures are suspect if the locations of mesh points are biased. Lattice meshes are particularly troublesome [20], as is geometric regularity arising from some adjustment procedures such as point repulsion [36] and centroidal Voronoi tesselation [24]. Strain and stress rates are independent of rotations, i.e., the physics are isotropic. For spring networks, mesh structure may affect the ability to model this isotropy and reproduce uniform elasticity, independent of fracture.

For computational science validation it may help to have multiple meshes with nearly identical global properties, but with local differences. Simulations results over all the meshes can be compared, to see if the results are dependent on mesh artifacts. Fracture simulations are dependent, but point location variablity is considered a subset of material property variability. Simulations over an esemble of meshes are collected to generate the range of possible experimental outcomes. Unbiased Poisson-disk sampling is ideal for these applications; a *maximal* distribution helps with angle bounds (Section 3.1) and performance [2].

The meshing literature abounds with methods for handling *sharp* features of the domain: small input angles, and edges

close to non-contained vertices. Spielman et al. [34] provides a parallelizable method for this in 3d. One key idea is to isolate and mesh these features before handling the rest of the domain [33; 32; 26]. The category of algorithms that recovers the domain boundaries before inserting interior points can be combined with this sort of preprocessing. For example, we may do that, as can Chew [8]. Preprocessing can even be combined with the family of methods that follows Ruppert's [29] variation that meshes (most of) the boundary at the same time as the interior. Cheng et al. [7] handles sharp features near vertices in 3d with a protective ball, and sharp angles between edges and facets are handled in the natural course of recovering the boundary. They isolate lower dimensional boundary features with protecting balls, similar to how we use the empty disks of our sampled boundary points. Handling input boundary facets in order of increasing dimension is common and effective. Fu [16] samples boundary vertices, curves, then surfaces, the same order as in our method. Fu's [16] anisotropic remeshing pipeline provides good triangles in practice, but without provable angle bounds. It includes random point insertion, but since it extends Dunbar and Humphreys's [12] advancing-front disk sampling method to surfaces embedded in 3d, and does Lloyd relaxation [25], points are geometrically biased.

We use a maximal Poisson-disk sampler as a subroutine. While we prefer our earlier method [13] for generating sample points, others may be used. White et al. [40], and Gamito and Maddock [17] are unbiased alternatives. Dunbar and Humphreys [12], and Wei [39] are very efficient.

Our triangulation algorithm is linear given sample points prelocated in a uniform grid of squares. Many problems, including DT, are known to be $O(n)$ when the points are sorted or otherwise geometrically organized [11; 5]. Some triangulation methods use a quadtree [18] or other background mesh to aid point location. Buchin and Mulzer [5] use quadtrees to control point insertion, yielding a linear Delaunay algorithm even over badly-distributed points. Kil and Amenta [21] provide a robust parallel-GPU implementation of a serial linear-time local Delaunay triangulation of nicely-distributed data. This is very similar to our setting. Kil and Amenta do some CPU pre-processing while our GPU version does none.

1.1. *Our Contribution*

We present a linear CDT algorithm based on uniform random points. Algorithm complexity and provable triangle angles are similar to the best known for (biased) uniform Delaunay refinement CDT methods. To our knowledge [1], ours is the first provably optimal algorithm based on Poisson-disk sampling. Efficiency is achieved using locality (background grid and bounded edge lengths) and radial sorting.

Section 3.2 and Remark 7 shows the near-equivalence of Delaunay refinement and our MPS approach in terms of the outcome, despite the processes being opposite. Delaunay refinement inserts points to reduce circumcircle radii and as a byproduct provably produces a (biased) maximal sampling. Our method produces a maximal sampling, and as a byproduct a provably good Delaunay mesh results. Our sampling is unbiased, but any maximal disk sampling is sufficient for provable angle bounds. For example, Section 3.2 may be a way to show that many practical Poisson-disk sampling methods, and many variants of Delaunay refinement, achieve provable angle bounds similar to Chew, because they achieve a maximal empty-disk sampling.

Our implementation performance appears reasonable compared to Shewchuk's [30; 33] popular Triangle software. Our typical serial running time is 2.7 seconds to triangulate 1,000,000 points, plus 10 seconds to generate the sample points. Our algorithm and Triangle take about the same amount of time to triangulate. Triangle generates points much faster, because the points it generates are deterministic rather than unbiased random. Our GPU triangulation code is a 2× speedup over our serial code, about 29% of the theoretical memory-limited speedup.

Our algorithm is modular in the sense that it may be incorporated into a complete mesh generation toolkit that performs many other steps, such as preprocessing sharp boundary features.

2. Algorithm

- **Preprocess** sharp boundary features.
- **Protect** the boundary of the domain with random disks.
- **Sample** the interior of the domain, until the set of disks is maximal.
- **Triangulate** the sample points.

For clarity, we describe sampling before protecting. We analyze the algorithm in Section 3; then give implementation details for triangulating serially (Section 4) and on the GPU (Section 5).

2.1. *Maximal Poisson Sampling*

Maximal Poisson-disk sampling selects random points $\{x_i\} = X$, from a domain, $\mathcal{D}$. The disk of radius r for each point contains no other points. x_i is chosen from $\mathcal{D}_{i-1}$, the remaining disk-free area of $\mathcal{D}$, without preference: the probability P of selecting a point from a subregion Ω is proportional to Ω's area. The maximal condition means that the points' disks cover the whole domain and no more points can be sampled.

$$\text{Bias-free: } \forall \Omega \subset \mathcal{D}_{i-1} : P\left(x_i \in \Omega\right) = \frac{\text{Area}(\Omega)}{\text{Area}(\mathcal{D}_{i-1})} \quad \text{(1a)}$$

$$\text{Empty disk: } \forall x_i, x_j \in X, i \neq j : ||x_i - x_j|| \geq r \quad \text{(1b)}$$

$$\text{Maximal: } \forall p \in \mathcal{D}, \exists x_i \in X : ||p - x_i|| < r \quad \text{(1c)}$$

In recent work [13], we have shown how to efficiently produce a sampling satisfying all three criteria. The main datastructure is a background uniform cell (square) decomposition to keep track of the remaining uncovered area of the domain. The diagonal of a square is r, so it can contain at most one dart.

We have two phases. In the first phase, darts (vertices, disk centers) are thrown into empty cells. If a new dart violates (1b), it is rejected. We only need to check a constant number of nearby cells. After a linear number of dart throws, the remaining uncovered area is expected to be small, and we switch. In the second phase, the main innovation is to build a polygonal approximation to the disk-free voids within cells. We weight each polygonal void by its area. We randomly throw darts based on these weights, which is the only non-linear step. Careful attention to placing or rejecting darts within the polygonal approximations leads to an unbiased sampling. Efficiency arises from careful handling of when to update and recalculate weights. The expected run-time is $O(n \log n)$; the $\log n$ dependence is very mild. The memory is deterministic $O(n)$. The number of cells $|\mathcal{C}| = \Theta(n)$.

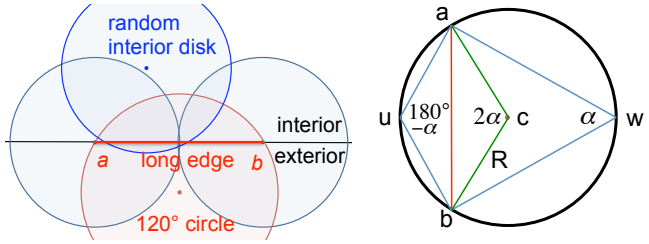
For this paper we treat that algorithm as a black-box that produces both the sample points and the cells containing them. We also rely on the ability of the black-box to accept some prescribed sample points on the domain boundary, and then generate the rest of the points needed to achieve a maximal distribution.

2.2. *Preprocess sharp features*

We assume sharp vertices have been protected by preprocessing using one of the methods from the introduction. Whether a vertex is too close to a non-containing edge and must be preprocessed, i.e. is sharp, depends on r. After preprocessing, what we require is that r is smaller than any input edge; and r is small enough that when we protect the domain boundary the disks for one input edge will not intersect another input edge, except perhaps for the disk centered at the common vertex of the edges.

2.3. *Protecting the domain boundary*

Pure maximal Poisson sampling [13] may introduce vertices arbitrarily close to the domain boundary. This poses no problems for maximal Poisson sampling per se, but would result in triangles with unbounded small and large angles.



(a) Interior-disk. (b) Central Angle Theorem.

Fig. 3. (a) A dark-blue random disk covering the interior intersection point of the blue circles covers the forbidden red region.

To prevent this, we *protect* the domain boundary by introducing sample points exactly on the boundary, or at least some distance from the boundary. The disks of these samples cover a neighborhood of the boundary, preventing the introduction of points that could create triangles with bad angles. The price is introducing a sample bias near the boundary.

We follow the simple but effective methods of Chew [8]. The main idea is to place a single dart at each vertex, and then protect edges by maximally placing random darts along them.

It is easy to space disks between r and $2r$ apart. However, disks between $\sqrt{3}r$ and $2r$ apart do not overlap enough to protect the boundary. We have two solutions: *close-disks* and *interior-disks*.

2.3.1. *Protecting with close-disks*

It is also easy to space boundary disks between $\sqrt{3}r/2$ and $\sqrt{3}r$ apart. This results in some boundary r-disks containing each other's centers, and an angle lower bound of $\arcsin \sqrt{3}/4 \approx 25.6°$ instead of 30°. The quality and timing results given for the implementation are for this strategy.

2.3.2. *Protecting with interior-disks*

See Figure 3(a). We may preserve the property that r-disks do not contain each other's centers and obtain a 30° angle bound by following the approach of Chew [8]: protect long ($> \sqrt{3}r$) edges by introducing a disk centered in the interior. Let a and b be consecutive samples on a boundary edge, and C_a and C_b their r-radius disks. Consider a circle with chord $\overline{ab}$. The Central Angle Theorem, Figure 3(b), says the chord subtends the same angle for any point on the arc of the circle on one side of $\overline{ab}$. The angle only depends on the circle radius. Any point inside the circle makes an even larger angle with $\overline{ab}$. After protecting the boundary, only interior points are added; we will generate a covering triangulation of the protected boundary. The angle that a constraint edge makes with a visible point is a lower bound on the maximum angle in any covering triangulation, regardless of additional points or choice of triangulation edges [28]. So we cannot place any samples inside the

120° circle for $\overline{ab}$, $C_{120°}$. Part of $C_{120°}$ is already covered by C_a and C_b, and we seek to cover the remainder with an interior disk.

A natural choice is centering a disk at the midpoint of the arc $\overset{\frown}{ab}$ of $C_{120°}$; instead of $C_{120°}$ Chew uses the circle of points making 90° angles with $\overline{ab}$. We actually have a lot more freedom than this; see Figure 3(a). Circles C_a and C_b intersect exactly once on the domain-interior side of $\overline{ab}$. We show in Section 3.3 that it is sufficient to place a disk anywhere covering that intersection point, as long as the disk center is outside C_a, C_b and $C_{120°}$. Such a disk covers the remaining forbidden region inside $C_{120°}$. Random disk placement reduces the bias, and improves fracture mechanics simulations by varying the angle between boundary and interior edges.

2.4. *Triangulate*

Our cell structure enables a local, simple, and fast algorithm for constructing the constrained Delaunay triangulation, CDT. We shall see in Section 3 that the length of CDT edges are bounded by $2r$. This ensures that for any point, the other points of its star are all in nearby cells. The number of nearby cells is constant, so the number of sample points they contain are bounded by that constant. Thus, we may construct the star of every point in constant time, using any of the standard constrained Delaunay algorithms, in linear total time.

While divide and conquer is known to be the fastest serial algorithm, for effective parallelism we use an algorithm based on the locality of points. Radial sorting is particularly efficient on GPUs. Our square background grid helps with efficiency. Details are given starting in Section 4.

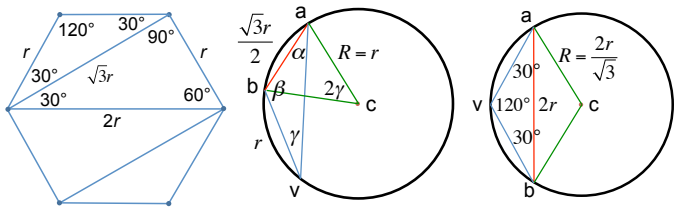
3. Analysis of the algorithm

3.1. *Angle bounds*

Here we show that the algorithm produces triangle angles between 30° and 120°, or 25.6° and 124.4°. The results are nearly the same as Chew [8] but the proofs, i.e. the reasons the results hold, are reversed. We ignore triangles containing the vertex common to two input edges or other sharp vertices, as their quality depends on the details of the preprocessing step.

Definition exterior- interior-center. A CDT $T = \triangle v_1v_2v_3$ with circumcenter c is an *exterior-center triangle* iff $\overline{v_1c}$ crosses $\overline{v_2v_3}$ and a constraint edge $\overline{ab}$, and is an *interior-center triangle* otherwise. C is the circumcircle and R the circumradius.

Lemma 1 *Interior-center triangles have $R \le r$.*



(a) Interior △. (b) Min-angle on bdy. (c) Max-R on bdy.

Fig. 4. (a) Extreme cases for interior triangles. (b) Boundary $\triangle vab$ with $|\overline{ab}| = \sqrt{3}r/2$, $|\overline{bv}| = r$, and $R = r$: $\angle v = \gamma = \arcsin\sqrt{3}/4$; $\alpha = \angle a = 30°$; $\beta = \angle b = 150° - \arcsin\sqrt{3}/4 < 124.4°$. (c) Boundary $\triangle vab$ with $|\overline{ab}| = 2r$, $\angle v = 120°$ and $R = 2r/\sqrt{3}$.

Proof $c \in \mathcal{D}$ so c is covered by radius-r Poisson disks. $\{v_i\}$ are its closest visible points. □

Lemma 2 *Exterior-center triangles have $R \le r$ (close-disks) or $R \le 2r/\sqrt{3}$ (interior-disks).*

Proof No constraint edge can cross an edge of T. No constraint edge vertex can be in C and visible to v_1. Hence ∃ constraint edge $\overline{ab}$ crossing arc $\overset{\frown}{v_2v_3}$ twice and crossing $\overline{v_1c}$, with $\overline{ab} \cap C$ visible to v_1. By the Central Angle Theorem, $\angle acb = 2(180° - \angle av_1b)$. $\angle av_1b \le 120°$ by construction, hence $\angle acb \ge 120°$, and $R = |\overline{v_1c}| \le |\overline{ab}| \cot 60°$. □

Lemma 3 *CDT edges e have length $r \le |e| \le 2r$ (interior-disks) or $\sqrt{3}r/2 \le |e| \le 2r$ (close-disks).*

Proof No visible sample points are closer than r or $\sqrt{3}r/2$ to each other. For an interior-center triangle, edge lengths are at most the circumcircle diameter. For an exterior-center triangle, $\overline{v_2v_3}$ is its longest edge and $|\overline{v_2v_3}| \le |\overline{ab}|$. □

These Lemmas are nearly the same as the conclusion of Chew's [8] Theorem 1, although the construction leading to the proof of the circumcircle radius condition is different: Chew's algorithm inserts the centers of large circles leading to a tight packing of points, while we insert tightly-packed points leading to no large circles.

Chew's Corollary to his Theorem 1 follows for the interior-disks strategy.

Corollary 4 (Chew's Corollary) *(1) CDT angles are between 30° and 120°. (2) $|e| \le 2r$.*

Corollary 5 (close-disk angles) *For the close-disks strategy CDT angles are between $\arcsin\sqrt{3}/4 > 25.6°$ and $150° - \arcsin\sqrt{3}/4 < 124.4°$.*

Figures 3(b) and 4 outline the proofs and show that the angle and edge length bounds are tight. Recall we exclude the analysis of triangles with two constraint edges. For interior-disks, the smallest circumradius occurs for a triangle with edge lengths r-r-r: $R \ge r/\sqrt{3}$; for close-disks r-r-$\sqrt{3}r/2$ gives $R \ge 2r/\sqrt{13}$.

strategy	angle		edge length		circumradius	
	min	max	min	max	min	max
interior-disks	30°	120°	r	$2r$	$0.577r$	$1.155r$
close-disks	25.4°	124.4°	$0.866r$	$2r$	$0.555r$	r

3.2. *Triangulations and maximal samplings*

We observe the near-equivalence of maximal disk samplings and good triangulations. An epsilon-net is a point set satisfying (1c); see Haussler and Welzl [19] for a full definition.

Lemma 6 *The vertices of any triangulation with bounded circumcircle radii $\leq r$ forms an epsilon-net for circles with $\epsilon = r$, regardless of Delaunay property, angle bounds or smallest edge length.*

Proof If the triangulation is Delaunay, then this follows from Delaunay-Voronoi duality: Delaunay circle centers are the vertices of convex Voronoi cells. But the Delaunay property is not required, as the Voronoi diagram for a single triangle shows that every point p inside $\triangle xyz$ is within distance r of one of its vertices. □

Remark 7 *If additionally all vertices are at least r from each other, then the vertices are a maximal sampling with radius r satisfying properties* (1b) *and* (1c).

A Delaunay triangulation with all edge lengths at least r has this property, because the nearest neighbor graph is a subgraph of the DT edge graph. In particular, the points of Chew's algorithm (and variants) are a maximal sampling.

3.3. *Geometry of Protecting the Boundary*

Here we prove that we have a lot of freedom in placing interior-disks protecting the boundary. Let a and b be consecutive boundary-edge samples, and label the construction as in Figure 5. Let n be the interior point of intersection between C_a and C_b. Let region F be the points in the domain outside C_a and C_b making angles 120° or greater with $\overline{ab}$. $\angle anb = \alpha > 120°$ because we only protect "long" edges, those with $|ab| > \sqrt{3}r$.

Theorem 8 *Any r-disk C_x covering n covers all of F.*

Proof Let point $q = C_b \cap C_{120°}$ and point $t = C_a \cap C_{120°}$ on the interior side of $\overline{ab}$. WLOG assume x is closer to a than b, hence closer to t than q. The lemmas that follow prove that x is also closer to q than n. The ideas are that x lies above $\overrightarrow{bt}$, which in turn lies above the perpendicular bisector of $\overline{nq}$. ("Above" means on the opposite side of the line as a.) So all the vertices n, q, and t of F lie inside C_x.

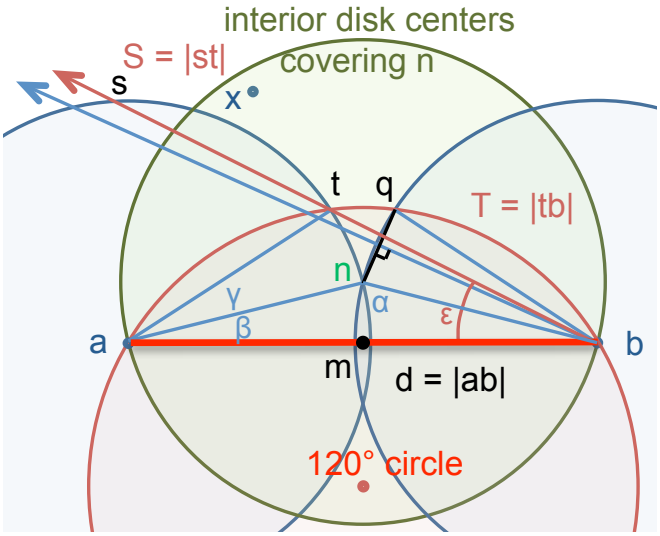


Fig. 5. Any disk C_x covering the interior intersection point $n = C_a \cap C_b$ protects the boundary, as long as x is outside C_a, C_b and the 120° circle of $\overline{ab}$. Angles and distances show that x is closer to q and t than n, so C_x covers forbidden region $F = \triangle ntq$.

One side of F is non-convex, but C_x intersects $C_{120°}$ once between $\overset{\frown}{qb}$ and once between $\overset{\frown}{ta}$, and circles intersect at most twice, so all of F lies inside C_x. □

We first show $\overrightarrow{bt}$ lies above $\perp \overline{nq}$.

Lemma 9 $\epsilon \geq \beta + \gamma/2$.

Proof Here $\alpha = \angle anb$, $\beta = \angle nba = \angle nab$, $\gamma = \angle tan = \angle qbn$, and $\epsilon = \angle abt = \angle baq$.

Since $\triangle anb$ is isosceles $\beta = 90° - \alpha/2$. Since $\triangle atb$ is a triangle and $\angle atb = 120°$, $\beta + \gamma = 60° - \epsilon$. Linearly combining these gives $\beta + \gamma/2 = 75° - \alpha/4 - \epsilon/2$. So $\beta + \gamma/2 \leq \epsilon$ is equivalent to $50° - \alpha/6 \leq \epsilon$. Since $120° \leq \alpha \leq 180°$ and $0° \leq \epsilon \leq 90°$, we may take the sines of each side. It remains to check $\sin(50° - \alpha/6) \leq \sin\epsilon$.

By the law of sines over $\triangle tab$, $\sin\epsilon = r\sin(120°)/d$, where $d = |ab|$. Considering right triangle $\triangle nma$, we have $\sin(\alpha/2) = d/2r$. So $\sin\epsilon = \sin(120°)/(2\sin(\alpha/2))$. Our check reduces to $\sin(50° - \alpha/6) \leq \sin(120°)/(2\sin(\alpha/2))$.

Let $h(\alpha) = \sin(\alpha/2)\sin(50° - \alpha/6)$. Our check is $h(\alpha) \leq \sin(120°)/2$. Equality holds at $\alpha = 120°$. We now show that $h' \leq 0$ for our range of α. $h' = \cos(\alpha/2)\sin(50° - \alpha/6)/2 - \sin(\alpha/2)\cos(50° - \alpha/6)/6$. So $h' \leq 0 \iff \tan(50° - \alpha/6) \leq \tan(\alpha/2)/3$. For $\alpha \in [120°, 180°]$, the right hand side is decreasing in α and the left hand side increasing. □

Corollary 10 *Any point x above $\overrightarrow{bt}$ is closer to q than n.*

We now show that x lies above $\overrightarrow{bt}$. Partition space by the perpendicular to $\overline{ab}$ through t: halfspace T_b contains b and $a \in T_a$. Points in T_b below the ray are inside $C_{120°}$. The next lemma shows that points in T_a below the ray are either inside C_a or too far from n.

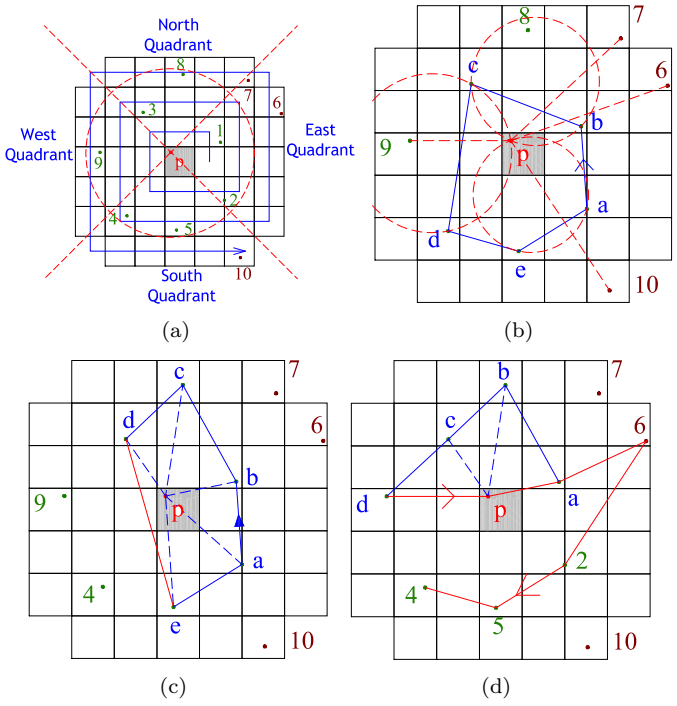


Fig. 6. Constructing the local CDT for p. (a) Points sharing an edge with p must lie within a 7×7 template. Cells are ordered in a *spiral* according to the blue curve. Points are numbered by the order they are considered in Step i. (b–d) Alternative configurations after removing points of $\mathcal{P}$ based on boundary constraints, and Delaunay circles. Letters indicate the order of points in $\mathcal{P}$. (b) No constraints. (c) A nearby boundary. (d) p lies on the boundary.

Lemma 11 *$|ts| = r$, where $\{t, s\} = \overrightarrow{bt} \cap C_a$.*

Proof Label the construction as in Figure 5. Let $T = |bt|$ and $S = |ts|$. Recall that $|ab| = d$ and C_a has radius r. By the power-of-a-point theorem, $(T + S)T = (d + r)(d - r)$, so $S = (d^2 - r^2)/T - T$. By the law of cosines for $\triangle abt$, $d^2 = T^2 + r^2 + Tr$. Combining these gives $S = r$. □

Corollary 12 *Any point x within r of n is above $\overrightarrow{bt}$.*

4. Local CDT

Our CDT algorithm iterates over each point p of the maximal Poisson distribution, constructing its star, i.e. the triangles containing it. Define the *CDT-star* as the triangles containing p in a true constrained Delaunay triangulation of the entire point set; we ensure our constructed star is a CDT-star. Our background grid and GPU considerations encourage a sorted-angle approach. Points in nearby cells are our candidates $\mathcal{Q}$ for the star. They are inserted into $\mathcal{P}$, the vertices of the star in clockwise order around p.

Our algorithm takes three passes. In the first pass points from $\mathcal{Q}$ are added to $\mathcal{P}$. After this pass $\mathcal{P}$ contains all the CDT-star vertices, but may also contain some extra vertices. In the second pass these extra vertices are removed based on constraints or prior points' stars. In the third pass extra vertices are removed based on the Delaunay principle. Adding and removing vertices are implicit edge flips converting a star triangulation to a constrained Delaunay triangulation.

4.1. *Serial CDT algorithm details*

We provide these details for others wishing to reproduce or improve our results. Constrained Delaunay edges have length $\leq 2r$, so only points within $2r$ of p are relevant to p's CDT-star. (We shall see below that a visibility-blocking constraint edge will have at least one vertex within $2r$ as well.) These points lie within a 7×7 template of squares (with corners removed), centered at the cell of p. In the first pass we gather and sort these points q by angle around p. To avoid any expensive square-roots, we use the slope of $\overline{pq}$ as a surrogate for the angle. The geometric centers of the cells of the template have a fixed sorted-order around, and distance to, the center cell. This defines a spiral ordering to the cells, which speeds sorting the sample points and other checks.

Terminology. An edge that is known to exist in the CDT-star, either by constraints or by a prior star calculation, is a *validated* edge. An edge that is known to not exist, by a prior star calculation, is an *invalidated* edge. An edge that may exist is a *candidate* edge, and is neither validated (constrained) nor invalidated. We often loop over the points of $\mathcal{P}$, sometimes circularly; let p_0 be the current point, p_- the prior point and p_+ the next point in $\mathcal{P}$.

(i) Populate $\mathcal{P}$ with points of $\mathcal{Q}$ sorted by angle around p. Visit the cells of the spiral in order; see Figure 6(a). When a cell contains a point q_i,

  (a) If $|\overline{pq_i}| > 2r$, discard q_i.

  (b) Find the quadrant containing q_i. Insert q_i into the quadrant's list of points, sorted by the slope of $\overrightarrow{pq_i}$. If two rays have identical slope, e.g. both of them lie on an input edge, discard the point farther from p.

  (c) If q_i contains a constrained edge, ensure its other vertex a is added to a quadrant, even if a is farther than $2r$ from p. (This edge may be used in Step iia, and is needed for correctness in Section 4.2.)

  Concatenate the quadrants' sorted lists to form $\mathcal{P}$.

(ii) Remove vertices of $\mathcal{P}$ based on (in)validated edges. Consider each $p_0 \in \mathcal{P}$.

  (a) If $\overline{p_0 p_k}$ is validated but p_0 and p_k are not consecutive in $\mathcal{P}$, then remove p_j, $0 < j < k$, whenever $\overline{pp_j}$ crosses $\overline{p_0 p_k}$.

  (b) If $\overline{p_0 p}$ is invalidated, remove p_0. Mark any validated $\overline{pp_0}$.

(c) Remove domain boundaries in connected components of the domain intersected with the radius $2r$ circle centered at p other than p's component: if p_0 lies on a one-sided domain boundary, and p is in the exterior, remove p_0. Here the exterior is defined locally by the cone of the two consecutive boundary edges containing p_0.

Checking neighboring stars is relatively expensive because it involves chasing a lot of edges.

(iii) Remove vertices of $\mathcal{P}$ based on geometry. Check Delaunay at p_0.

(a) If $\overline{p_0 p}$ is a candidate edge and p_0 is located outside the circumcircle of $\triangle p_- p_+ p$, remove p_0 and recheck p_-. (Incircle test, edge flip.)

When p_0 is removed we must recursively recheck the prior point p_-. In the natural iteration we will check the subsequent point p_+. This corresponds to a chain of edge flips in the incremental Delaunay algorithm. We must keep checking until all consecutive triples of points have been checked and none removed.

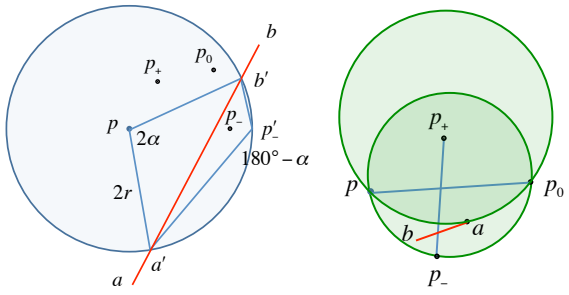
The star of p are validated edges when subsequently running the local CDT algorithm for nearby points. This breaks ties consistently for four or more cocircular points. Any other potential edge with p is invalidated.

4.1.1. *Implementation performance details*

A quadrant is defined by the two straight lines with slope ± 1 passing through p. Let δx be the x-coordinate of $q - p$ taken as vectors; δy is analogous. In Step i finding the quadrant takes only two subtractions, two absolute values, and two comparisons. Depending on the quadrant, we use $\delta x/\delta y$ or its reciprocal as the slope, and sort ascending or descending. Computing slope requires only one division. The number of points in a quadrant is small, so maintaining sorted order is fast. It might be possible to further optimize Step i. For example, the cells could be re-ordered so that the next point is more likely to be at the end of a sorted list. But as described the run-time of the first pass is already smaller than the other passes. For GPUs, sorting by angle is a fast primitive, so we use that instead of the quadrants and slopes.

In Step ii we check the N neighbors of p_0 for validated edges. This check dominates the run time. ($N \leq 14$ from the minimum angle in the CDT.) For the run-time of Step iii, we bound the number of Delaunay checks. There are $|\mathcal{P}| < 45$ initial triples of consecutive points. Every point that is removed generates two new triples that must be checked. So there are at most $3|\mathcal{P}|$ checks. Together these two steps take $O(N|\mathcal{P}|)$ time.

For Delaunay circle checks, we use Shewchuk's [31] "incircle" primitive. For determining when edges cross, instead of Shewchuk's "orientation" primitive we use our own based on triangle area because the known sorted order allows us a small shortcut. We use a fixed-length array for $\mathcal{P}$. To minimize memory movement, instead of actually removing points from the array, we mark them as "removed" using an ancillary array.



(a) Case 1 (b) Case 2

Fig. 7. Illustrations for the proof of Lemma 15.

4.2. *Correctness*

The main arguments behind the correctness of our CDT algorithm are that the constructed star is a CDT of the retained points, and no discarded point is in p's CDT-star (the star of p in a true CDT).

Theorem 13 *On termination, $\mathcal{P}$ are the vertices of the star of p in a CDT of the entire domain.*

Proof By Lemma 14 we start with a superset of the CDT-star of p. Lemma 15 shows that removed vertices are not in the CDT-star of p. Lemma 16 shows that the algorithm produces the CDT of the remaining vertices and constrained edges. This CDT is a proper subset of the CDT of the entire domain by the Delaunay principle, and by observing all relevant constraints. □

Lemma 14 *After Step i, $\mathcal{P}$ contains all the vertices of the CDT-star of p, plus perhaps some extra vertices.*

Proof Trivial. By Corollary 4, CDT edge lengths are at most $2r$, so only points inside the template are in the CDT-star of p. □

Lemma 15 *Removed vertices are not in p's CDT-star.*

Proof Any p_0 discarded by constraints in Step ii is obviously not in the CDT. It remains to consider discarding p_0 based on geometric criteria in Step iii. p_0 is discarded iff $\overline{pp_0}$ is not in the Delaunay triangulation of p_-, p_0, p_+ and p, and $\overline{pp_0}$ is not validated. The only way $\overline{pp_0}$ could be in a CDT but not a DT is if some constraint edge passes through $\triangle pp_-p_+$. WLOG we must consider two cases, the possibility of a constraint edge $\overline{ab}$ crossing (1) $\overline{pp_-}$ and (2) $\overline{p_-p_+}$ but not $\overline{pp_-}$.

Case 1: Consider the $2r$-circle centered at p, $2C_p$. If a or b is in $2C_p$, then Step ii would have removed p_- before Step iii. Otherwise, show next that angle and edge length bounds imply the existence of some other vertex in $2C_p$ with constraint edges that would have removed p_-.

See Figure 7 and Mitchell [28]. Let $\overline{ab}$ be the closest constraint edge to p_- crossing $\overline{pp_-}$ with no vertex in $2C_p$. Let a' and b' be the intersection of $\overline{ab}$ with $2C_p$, and p'_- the intersection of $\overrightarrow{pp_-}$ with $2C_p$.

By the Central Angle Theorem, $\angle a'pb' = 2\alpha$ and $\angle a'p'_-b' = 180° - \alpha$. $\alpha = \arcsin(|\overline{a'b'}|/(4r)) < \arcsin(|\overline{ab}|/(4r))$. By Corollary 4 $|\overline{ab}| < 2r$, so $\angle a'p'_-b'$ exceeds 120°. (Regardless of using the close-disks or interior-disks strategy.)

Let v be the vertex in $\triangle p_-a'b'$ closest to $\overline{a'b'}$; perhaps $v = p_-$. Consider the CDT of the **convex hull** of the input domain, which is a proper superset of the CDT of the domain. In this larger triangulation, consider the triangle U containing v and a subset of $\overrightarrow{vp}$; perhaps $U = \triangle avb$. U has angle at v in excess of 120°, and so must be exterior to the domain. Since $\overline{ab}$ was chosen to be the closest constraint edge, no constraint edge crosses $\overrightarrow{vp}$ inside $\triangle avb$. So v is on the boundary of the domain, and its constrained domain-boundary edges would have removed p_- in Step ii.

Case 2: The remaining case is $\overline{ab}$ crosses $\overline{p_-p_+}$ but no constraint edge crosses $\overline{pp_-}$ or $\overline{pp_+}$. One of a or b lies inside $\triangle pp_-p_+$. If $\overline{ab}$ crosses $\overline{pp_0}$ then $\overline{pp_0}$ is not in the CDT; otherwise WLOG a is the closest point visible to all of $\overline{pp_0}$, and a lies inside $\triangle pp_-p_0$. The Delaunay check showed that the circumcircle of $\triangle pp_-p_0$ contained p_+, hence the circumcircle of $\triangle pap_0$ also contains p_+. Since no constraint edge crosses $\overline{pp_+}$ or $\overline{pp_0}$, let c be the vertex in $\triangle pp_+p_0$ closest to and visible to $\overline{pp_0}$; c might be p_+. Then a, c, p_0, and p are mutually visible, and the circumcircle of $\triangle pap_0$ contains c, which invalidates $\overline{pp_0}$. □

Lemma 16 (clean-up is flipping) *After Step iii, $\mathcal{P}$ defines the CDT of the vertices of $\mathcal{P} \cup p$ and any constraints.*

Proof Constraints: All constraint edges between p and p_0 are respected due to Step iib. All constraint edges between p_0 and some other vertex are respected due to Step iia. At the start of Step iii all points of $\mathcal{P}$ are visible to p. Also, consecutive points of $\mathcal{P}$ are visible to each other: no constraint edge blocks them because any such edge either has vertices that would make them non-consecutive, or remove one of them in Step ii. Visibility between consecutive points is invariant during Step iii because any p_0 with a constraint edge crossing $\overline{p_-p_+}$ is inside $\triangle pp_-p_+$ and will not be removed by a Delaunay check.

Delaunay: Delaunay's Theorem states that satisfying the empty-circle property between pairs of triangles sharing an edge is equivalent to satisfying the empty circle property globally. Based on this, a standard Delaunay algorithm is to take any triangulation and then flip edges to satisfy the local Delaunay criteria, recursively checking the new adjacent triangle pairs.

Removing vertex p_0 from the star corresponds to flipping $\overline{pp_0}$ with $\overline{p_-p_+}$. Our algorithm flips recursively through the star by iteration and backtracking on index j of p_0. The only difference between our algorithm and the standard flip algorithm is that we discard p_0. This is acceptable because we are only constructing the star of p, and a property of the standard algorithm is that if a flip removes an edge, there is a witness that the removed edge is non-Delaunay, so no subsequent flip will every re-introduce that edge. □

5. GPU implementation

We implemented a GPU version of the CDT algorithm, and also our black-box that generates the random points [13]. We ran it on points in a square domain, which is typical for fracture simulations. The localization provided by grid cells is central to our parallelization.

The CDT for p depends on three layers of cells, a 7×7 grid (with corners removed) centered at the cell of p. However, when generating the CDT we only change the data-structures associated with the center cell. We may simultaneously work on two 7×7 grids as long as they do not contain each others' center cell. Active cell centers are offset from one another by multiples of 4 in x and y indices; there are no race conditions between threads. Each thread begins with a 4×4 grid of center cells. We use global synchronization after each update; the cost is equivalent to a kernel relaunch, and has minimal overhead. There are sixteen stages, so that every cell (every p) is the center of a 7×7 grid in one stage. Since each thread can now be considered independently, each thread can imitate the serial algorithm; see Section 4.

Load balancing is achieved by having many more threads than processors, at least for the domains and mesh sizes of interest. The amount of work a given thread does at each stage varies widely; e.g. threads with empty cells return immediately. However, each processor works on many threads between global synchronization stages. In particular, there are enough memory requests to keep the DRAM controllers busy. We also considered atomics, but that approach appeared more complicated, and may perform worse.

Recall we use an alternative to Step i because GPUs are very fast at calculating angles due to their specialized hardware for transcendental functions. We dispense with quadrants and build one list directly. We gather the points from the 7×7 grid with distance to $p \leq 2r$ and add them to $\mathcal{P}$. We visit the cells in the angular order of their center point; this results in points being in nearly-sorted order. (This is different than the spiral order.) Then we run a local insertion sort. The nearly-sorted order reduces data movement.

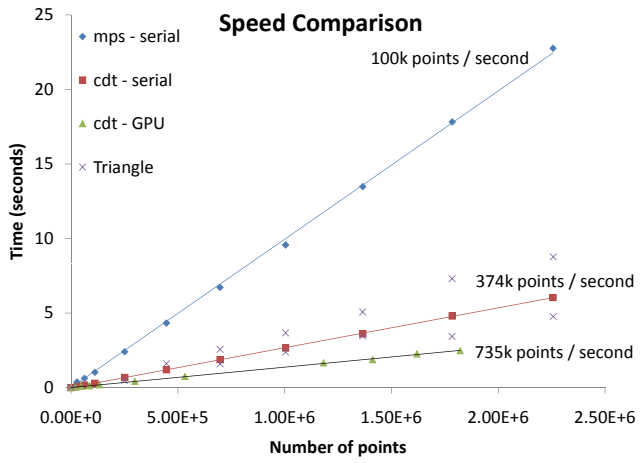


Fig. 8. The data points are for runs at different resolutions r. For a given resolution, we plot Triangle's maximum and minimum times. Our serial CDT is competitive with Triangle's median times. Our GPU CDT is about a 2× speedup over our serial CDT; the GPU memory bandwidth is 6.7× the CPU's. Our serial and GPU CDT implementations show near-linear performance. Our method and implementation for generating maximal Poisson samplings (MPS) is competitive with the best for MPS; the MPS family of methods is slower than Delaunay refinement for generating points.

If two points have the same angle, we remove the one farther from p.

The result is the same as the serial version, except that we did not collect constraint edges if one of its vertices is farther than $2r$ from p. We gather and sort any far constraint edges on the fly in Step ii. This reduces total memory access because edges are only considered once, in Step ii and not in Step i. The remaining GPU and serial steps are the same.

6. Implementation Performance

6.1. *Run-time and memory*

The serial implementation was tested on a laptop.[1] We ran the code over four typical fracture mechanics domains: roughly-square surface patches, some non-convex, with various combinations of holes and two-sided interior boundary edges. These differences had little effect on the run-time, memory, or mesh quality. As illustrated in Figure 8, we generated 100,000 random points/second and triangulated 370,000 points/second. The sampling density had little effect: both algorithms show a near-linear complexity in the number of mesh points, with only a very slight rise.

Figure 9 demonstrates serial memory usage. The phases of sampling and triangulating are visible. Phase II consumes more memory than Phase I, because the geometric voids are

[1] 2010 vintage. Intel® Core™ i7-620M at 2.67 GHz, 4 MB cache; 4 GB RAM; 64-bit Windows 7 OS.

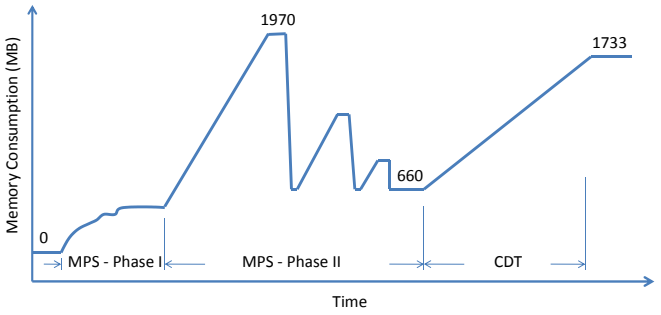


Fig. 9. CPU memory use while generating a 8,271,560 point mesh.

more complicated than simple squares. However, the difference between Phase II memory and the memory needed to store the final output is relatively small.

6.1.1. *GPU performance*

We ran our GPU CDT algorithm on an NVIDIA™ GeForce™ GTX 460 with 336 CUDA Cores and 1 GB GDDR5 RAM. We triangulated 735,000 points/s, about a 2× speedup over the serial code; see Figure 8. Since memory bandwidth is the limiting factor, the theoretical best speedup is about 6.7×. The memory bandwidth is 17.1 GB/s for CPU, and 115 GB/s for GPU; the ratio is 6.7. The laptop CPU can perform 24.5 GFlops/s in turbo mode, the GPU's max is 907 GFlops/s; the ratio is 37. Due to GPU memory constraints, the largest mesh we could produce had about 2 million points.

6.1.2. *Comparison to Triangle*

In serial we typically triangulate 1,000,000 points in 2.7 seconds, plus 10 seconds to generate the sample points and grid. Our code is written in C++ and was compiled under Windows. We compiled the C code for Triangle [30; 33] under Linux, but on the same hardware. Given 1,000,000 random points in the unit square, Triangle took about the same amount of time as our code. Triangle has an internal reporting mechanism, and it reported between 2.4–3.7 seconds, frequently taking 2.7 seconds. We do not know why the times varied so much; the times reported by our code did not vary more than 5%. Triangle is faster when it generates its own point cloud; a local deterministic process vs. our global random one. There are differences in problem definition; the language, compiler and OS; scalability; and what constitutes difficult input; but this shows that our serial codes are close. Our GPU code is a 2× speedup over our serial code.

6.2. *Triangulation quality*

Poisson sampling leads to Delaunay triangulations with angles following a particular distribution [10; 23]. This distribution is fairly independent of the number of samples and

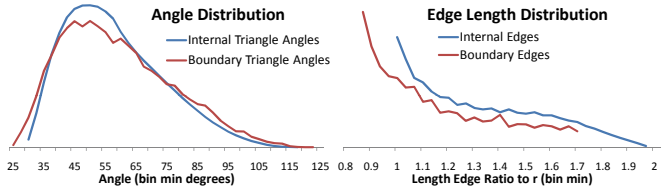


Fig. 10. Observed angles and edge lengths for a 1E6-point triangulation of the square. The angle histogram uses 2.5° bins. The edge-length histogram uses 1/30 width bins.

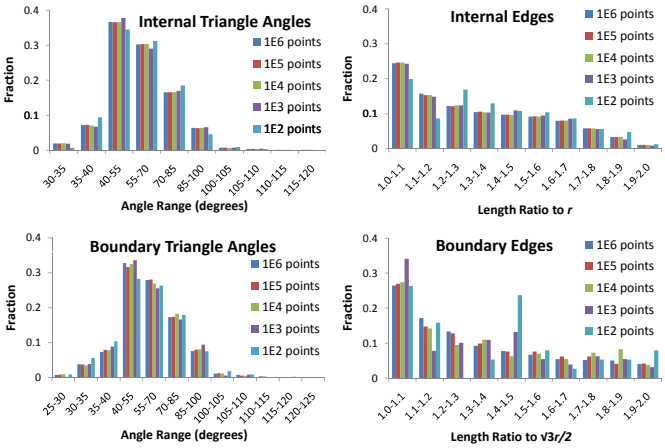


Fig. 11. Example distributions for the square, by number of points, n. About $4\sqrt{n}$ of the points are on the boundary. One sample run each. A *boundary edge* is a constrained edge. For any triangle containing a boundary edge, all three of its angles are *boundary triangle angles*; one of the angles may be at a vertex internal to the domain.

even the shape of the domain; see Figures 10 and 11. Most angles, 80%, lie between 40° and 80°, which is good for many applications. The edge length distribution also tends to be invariant; see Figure 11. The observed angle and edge extremes are consistent with the theoretical bounds. Other properties, such as edge-valence, have also been studied in the spatial statistics literature [15].

Tournois et al.'s Figure 2 [35] gives a plot of the angles for a typical Delaunay refinement algorithm, and refinement interleaved with smoothing. We see in Figure 10 that the distribution of angles in our random mesh has roughly the same shape as in Delaunay refinement. Theirs appears more jagged than ours, although this may be an artifact of the histogram widths or sample size.

7. Conclusion

In summary, we described a new method for generating a Conforming Delaunay Triangulation in two dimensions.

- Points are generated randomly, including points on the boundary. Point locations are unbiased, except near the boundary.
- Run-times are as good as the Delaunay refinement algorithms for triangulating, but not for generating points.
- Angles are as good as many Delaunay refinement algorithms.
- Points are generated in $O(n)$ space and $E(n \log n)$ time.
- Points are triangulated in $O(n)$ time and memory.
- The method is naturally parallelizable.

Random meshes can help in situations where mesh structure is a concern. Random meshes help validate the results of fracture propagation simulations, and avoid graphics rendering artifacts. The method works on planar straight-line graphs, including non-convex domains with internal 2-sided boundaries (cracks) and 1-sided boundaries (holes).

Empirical results are that the CDT can triangulate 374,000 points/second on a CPU, and 735,000 points/second on a GPU. The speed scales very well with the problem size. The output quality is largely invariant to the domain, and unconstrained angles are provably between 30° and 120°, or 25.6° and 124.4°.

We plan to extend the algorithm to higher dimensions. The current Poisson-disk sampling procedure is based on a constant disk radius; if this is relaxed, graded meshes that are unbiased may be possible.

Discrete algorithms are notoriously difficult to parallelize effectively. Their random memory access patterns (e.g. chasing chains of mesh edges) does not take good advantage of the hardware memory hierarchy. The GPU memory bandwidth is about 6.7× the CPU's; the 2× achieved speedup is a start. We intend to try exploiting locality in the 7×7 grids, perhaps using shared memory.

Acknowledgments

We thank Jonathan Shewchuk for discussing triangulations with us. We thank Joseph Bishop for discussing the effect of random meshes on simulation results with us. We thank the SIAM GD/SPM11 and SIGGRAPH 2011 reviewers for their helpful feedback. The Sandia authors thank the U.S. DOE, Office of Advanced Scientific Computing Research, SC-21, SciDAC-e, and Sandia's Computer Science Research Institute for supporting this work. The UC Davis authors thank the SciDAC Institute for Ultrascale Visualization, the National Science Foundation (grant # CCF-1017399), and the Intel Science and Technology Center for Visual Computing for supporting this work.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] Pierre Alliez, Giuliana Ucelli, Craig Gotsman, and Marco Attene. Recent advances in remeshing of surfaces. In Leila Floriani and Michela Spagnuolo, editors, *Shape Analysis and Structuring*, Mathematics and Visualization, pages 53–82. Springer Berlin Heidelberg, 2008. 10.1007/978-3-540-33265-7_2.

[2] Dominique Attali and Jean-Daniel Boissonnat. A linear bound on the complexity of the Delaunay triangulation of points on polyhedral surfaces. *Discrete Comput. Geom.*, 31(3):369–384, 2004.

[3] J.Ẽ. Bolander Jr. and S. Saito. Fracture analyses using spring networks with random geometry. *Engineering Fracture Mechanics*, 61:569–591, 1998.

[4] Lennart Bondesson and Jessica Fahlén. Mean and variance of vacancy for hard-core disc processes and applications. *Scandinavian Journal of Statistics*, 30(4):797–816, 2003.

[5] Kevin Buchin and Wolfgang Mulzer. Delaunay triangulations in $O(\text{sort}(n))$ time and more. In *50th Annual IEEE Symposium on Foundations of Computer Science*, pages 139–148, 2009.

[6] Siu-Wing Cheng, Tamal K. Dey, Herbert Edelsbrunner, Michael A. Facello, and Shang-Hua Teng. Sliver exudation. In *Proceedings of the Fifteenth Annual Symposium on Computational Geometry*, SCG '99, pages 1–13, 1999.

[7] Siu-Wing Cheng, Tamal K. Dey, Edgar A. Ramos, and Tathagata Ray. Quality meshing for polyhedra with small angles. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, pages 290–299, 2004.

[8] L. Paul Chew. Guaranteed-quality triangular meshes. Technical Report 89-983, Department of Computer Science, Cornell University, 1989.

[9] L. Paul Chew. Guaranteed-quality Delaunay meshing in 3D. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, pages 391–393. ACM, 1997.

[10] Richard Cowan. Some Delaunay circumdisks and related lunes. `http://www-personal.usyd.edu.au/~rcowan/professional/circum.pdf`, 2002.

[11] Olivier Devillers. Randomization yields simple $O(n \log^* n)$ algorithms for difficult $\Omega(n)$ problems. *International Journal of Computational Geometry and Applications*, 2(1):621–635, 1992.

[12] Daniel Dunbar and Greg Humphreys. A spatial data structure for fast Poisson-disk sample generation. *ACM Transactions on Graphics*, 25(3):503–508, July 2006.

[13] Mohamed S. Ebeida, Anjul Patney, Scott A. Mitchell, Andrew Davidson, Patrick M. Knupp, and John D. Owens. Efficient maximal Poisson-disk sampling. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2011)*, 30(4), August 2011.

[14] Herbert Edelsbrunner, Xiang-Yang Li, Gary Miller, Andreas Stathopoulos, Dafna Talmor, Shang-Hua Teng, Alper Üngör, and Noel Walkington. Smoothing and cleaning up slivers. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 273–277, 2000.

[15] Andreas Frey and Volker Schmidt. Marked point processes in the plane—a survey with applications to spatial modeling of communication networks. *Advances in Performance Analysis*, 1(1):65–110, 171–214, 1998. `http://www.mathematik.uni-ulm.de/stochastik/fundl/paper/frey/survey2/survey2.html`.

[16] Y. Fu and B. Zhou. Direct sampling on surfaces for high quality remeshing. *Computer Aided Geometric Design*, 26(6):711–723, 2009.

[17] Manuel N. Gamito and Steve C. Maddock. Accurate multidimensional Poisson-disk sampling. *ACM Transactions on Graphics*, 29(1):1–19, December 2009.

[18] Sariel Har-Peled and Alper Üngör. A time-optimal Delaunay refinement algorithm in two dimensions. In *Proceedings of the Twenty-First Annual Symposium on Computational geometry*, SCG '05, pages 228–236, 2005.

[19] D. Haussler and E. Welzl. ϵ-nets and simplex range queries. *Discrete and Comput. Geom.*, 2(1):127–151, 1987.

[20] Milan Jirásek and Zdenek P. Bazant. Particle model for quasibrittle fracture and its application to sea ice. *Journal of Engineering Mechanics*, 121:1016–1025, 1995.

[21] Yong J Kil and Nina Amenta. GPU-assisted surface reconstruction on locally-uniform samples. In *Proceedings of 17th International Meshing Roundtable*, pages 369–385, 2008.

[22] David B. Kirk and James Arvo. Unbiased sampling techniques for image synthesis. In *Computer Graphics (Proceedings of SIGGRAPH 91)*, pages 153–156, July 1991.

[23] Günter Last. Gamma distributions in Poisson Voronoi and hyperplane tessellations. *Spatial Network Models for Wireless Communications*, 2010. `http://www.newton.ac.uk/programmes/SCS/seminars/040614001.pdf`.

[24] Yang Liu, Wenping Wang, Bruno Lévy, Feng Sun, Dong-Ming Yan, Lin Lu, and ChengLei Yang. On centroidal Voronoi tessellation—energy smoothness and fast computation. *ACM Transactions on Graphics*, 28(4):101:1–101:17, August 2009.

[25] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[26] Gary L. Miller, Steven E. Pav, and Noel J. Walkington. When and why Ruppert's algorithm works. In *Proceedings of the 12th International Meshing Roundtable*, pages 91–102, Santa Fe, NM, September 2003. Sandia.

[27] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, and Noel Walkington. On the radius-edge condition in the control volume method. *SIAM J. on Numerical Analysis*, 36(6):1690–1708, 1999.

[28] Scott A. Mitchell. Finding a covering triangulation whose maximum angle is provably small. *Int. J. Comput. Geometry Appl.*, pages 5–20, 1997.

[29] Jim Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms*, 18(3):548–585, 1995.

[30] Jonathan Richard Shewchuk. Triangle: a two-dimensional quality mesh generator and Delaunay triangulator. `http://www.cs.cmu.edu/~quake/triangle.html`, 1996-2005.

[31] Jonathan Richard Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, 18(3):305–363, October 1997.

[32] Jonathan Richard Shewchuk. Mesh generation for domains with small angles. In *Proceedings of the 16th Annual Symposium on Computational Geometry*, pages 1–10. ACM, 2000.

[33] Jonathan Richard Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22:21–741, 2002.

[34] Daniel A. Spielman, Shang-Hua Teng, and Alper Üngör. Parallel Delaunay refinement: Algorithms and analyses. *Int. J. Comput. Geometry Appl.*, 17(1):1–30, 2007.

[35] Jane Tournois, Pierre Alliez, and Olivier Devillers. Interleaving Delaunay refinement and optimization for 2D triangle mesh generation. In *Proceedings of the 16th International Meshing Roundtable, IMR*, pages 83–101. Springer-Verlag, October 2007.

[36] Greg Turk. Re-tiling polygonal surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, pages 55–64, July 1992.

[37] Alper Üngör. Off-centers: A new type of Steiner points for computing size-optimal quality-guaranteed Delaunay triangulations. *Comput. Geom. Theory Appl.*, 42:109–118, February 2009.

[38] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for Monte Carlo rendering. In *Proceedings of SIGGRAPH 95*, pages 419–428, August 1995.

[39] Li-Yi Wei. Parallel Poisson disk sampling. *ACM Transactions on Graphics*, 27(3):1–20, August 2008.

[40] Kenric B. White, David Cline, and Parris K. Egbert. Poisson disk point sets by hierarchical dart throwing. In *RT '07: Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, pages 129–132, September 2007.